# NWERC 2020 presentation of practice solutions

# NWERC 2020 presentation of solutions

## NWERC 2020 Jury

- **Arnar Bjarni Arnarson**
  Reykjavík University
- **Per Austrin**
  KTH Royal Institute of Technology
- **Jeroen Bransen**
  Chordify
- **Alexander Dietsch**
  FAU Erlangen-Nürnberg
- **Ragnar Groot Koerkamp**
  ETH Zürich
- **Bjarki Ágúst Guðmundsson**
  Google
- **Nils Gustafsson**
  KTH Royal Institute of Technology
- **Timon Knigge**
  ETH Zürich
- **Robin Lee**
  Google
- **Pehr Söderman**
  Kattis
- **Jorke de Vlas**
  Utrecht University
- **Mees de Vries**
  University of Amsterdam
- **Paul Wild**
  FAU Erlangen-Nürnberg

## Big thanks to our test solvers

- **Bernhard Linn Hilmarsson**
  ETH Zürich
- **Tómas Ken Magnússon**
  Google
- **Ludo Pulles**
  Leiden University
- **Bergur Snorrason**
  University of Iceland
- **Tobias Werth**
  Google

# NWERC 2021 presentation of solutions

November 24, 2021

## NWERC 2021 Jury

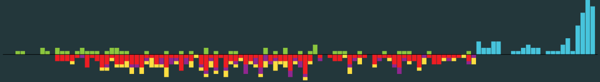
- **Per Austrin**
  KTH Royal Institute of Technology
- **Alexander Dietsch**
  e.solutions
- **Ragnar Groot Koerkamp**
  ETH Zurich
- **Antti Laaksonen**
  CSES
- **Bjarki Ágúst Guðmundsson**
  Google
- **Nils Gustafsson**
  KTH Royal Institute of Technology
- **Timon Knigge**
  ETH Zurich
- **Harry Smit**
  MPIM Bonn
- **Bergur Snorrason**
  University of Iceland
- **Pehr Söderman**
  Kattis
- **Jorke de Vlas**
  Utrecht University
- **Mees de Vries**
  IMC
- **Paul Wild**
  FAU Erlangen-Nürnberg
- **Michael Zündorf**
  Karlsruhe Institute of Technology

## Big thanks to our test solvers

- **Bernhard Linn Hilmarsson**
  ETH Zurich
- **Robin Lee**
  Google
- **Ludo Pulles**
  Leiden University
- **Johan Sannemo**
  Kattis
- **Reinier Schmiermann**
  Utrecht University
- **Tobias Werth**
  Google

# A: Atomic Energy

Problem Author: Jorke de Vlas



## Problem

Given are the '*explodification*' rules for an atom with a certain amount of neutrons:

- An atom with $k \leq n$ neutrons will be converted into $a_k$ units of energy.
- An atom with $k > n$ will be decomposed into parts $i, j \geq 1$ with $i + j = k$, which are then recursively *explodificated*.

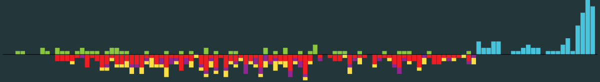
Given an atom with a fixed number of neutrons, what is the minimum energy released?

## Observations

Since the decomposition is arbitrary, we have to assume the worst case – for $k > n$ define:

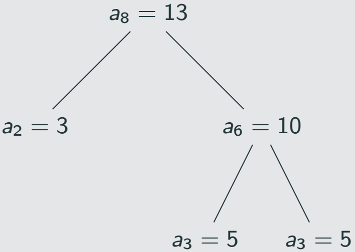
$$a_k := \min_{1 \leq i \leq k-1} a_i + a_{k-i}.$$

There are upto $10^5$ queries with $k$ upto $10^9$, so we cannot naively compute all values $a_i$ upto this maximum. Naive computation requires $O(k^2)$ time for the first $k$ values.



## Observation 1

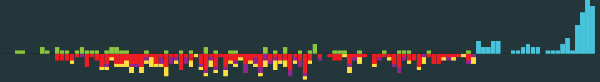
Our first crucial observation is that optimal solutions have a recursive structure. We can write any explodification sequence as a binary tree. This is the first sample, $k = 8$:




Recall this sample had $a_{1,\ldots,4} = \{2, 3, 5, 7\}$.

# A: Atomic Energy

Problem Author: Jorke de Vlas



## Observation 1

For a given query $k$, imagine recursively following the decomposition $a_k = a_i + a_{k-i}$ until we end up with a decomposition:

$$a_k = \sum_{j=1}^{m} a_{i_j} \quad \text{subj. to} \quad k = \sum_{j=1}^{m} i_j, \text{ with } i_j \in \{1, \ldots, n\}.$$

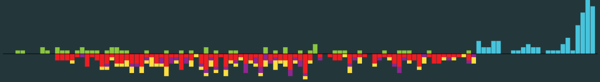
So the leaves of the decomposition tree are a collection of indices $i_j$ that sum to $k$. Is any decomposition $(i_j)$ satisfying the right hand side realizable?

No – to actually construct this explodification sequence we need to end with some $a_x, a_y$ with $x + y > n$. If $x + y \leq n$, there is no guarantee that $a_{x+y} = a_x + a_y$. (Example: for $n \gg 1$, a sequence of all $a_1$'s is generally impossible.)

A sequence is *realizable* if it contains two $x, y$ with $x + y > n$. After that, we can 'add' new atoms $a_{i_j}$ inductively to construct the explodification tree. In fact any 'prefix' of such a sequence is optimal.

# A: Atomic Energy

Problem Author: Jorke de Vlas

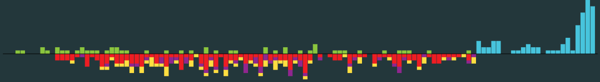


## Faster computation

Now we can improve the computation of the first $k$ values from $O(k^2)$ to $O(nk)$:

$$a_k = \min_{1 \leq i \leq n} a_i + a_{k-i}.$$

Of course this is still not fast enough with $k$ upto $10^9$.



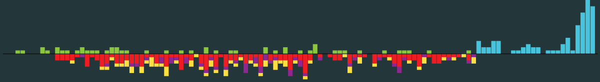
### Observation 2

Let $m \in \{1, \ldots, n\}$ minimize $a_m/m$. When a query $k$ is large enough, most of the terms in the decomposition will be $a_m$. Indeed, if after removing the two distinguished values $a_x, a_y$ from the sequence we still have $m$ or more values in the tree that are not $a_m$, by the pigeonhole principle there must be a subset of them that have indices that sum up to a multiple of $m$, and we can replace them by $a_m$'s to get a decomposition that is not worse.

Hence, any decomposition can be written in such a way that there are at most $m + 1$ terms that are not $a_m$. In fact we can rearrange the sequence to have these terms in the front, and then fill in the gap with $a_m$-terms.

# A: Atomic Energy

Problem Author: Jorke de Vlas



## Full solution

Let $m$ minimize $a_i/i$ over all $i \in \{1, \ldots, n\}$, and use the $O(nk)$ algorithm from earlier to construct the first $(m+1)n$ terms in time $O(n^3)$.

For each query $k$, find the smallest $j \geq 0$ such that $k - jm \in \{1, \ldots, (m+1)n\}$, and output with $a_{k-jm} + j \cdot a_m$.

Final runtime $O(n^3 + q)$. Efficient implementations of e.g. $O(n^4 + q)$ could also work.

Statistics: 421 submissions, 51 + ? accepted

# B: Bulldozer

Problem Author: Mees de Vries

## Problem

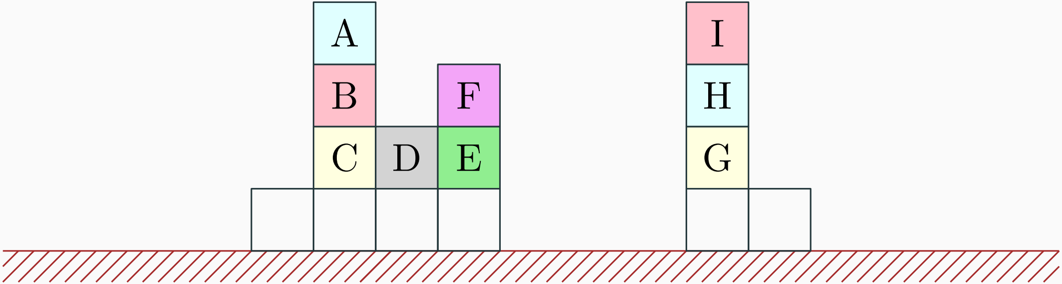
Given a row of stack of blocks, how many 'bulldoze' operations are needed to level all the blocks.

## Observations

- Each block can be 'buried' in two moves: push the bottom of the stack right, push the block left.
- It's never worse to do all burying operations at the end.
- All other blocks that start non-grounded end at an initially empty stack.
- Number the non-grounded blocks from left to right, where each stack is numbered bottom to top.
- The final solution has stretches of blocks that move left, stretches of blocks that move right, mixed with stretches of blocks that are buried.
- We have infinite space on the left and right, and the stretches of blocks that go there contain full stacks of blocks only.

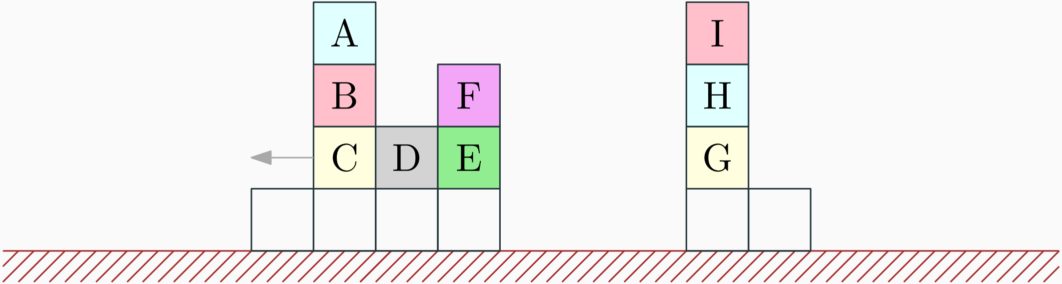
# B: Bulldozer

Problem Author: Mees de Vries

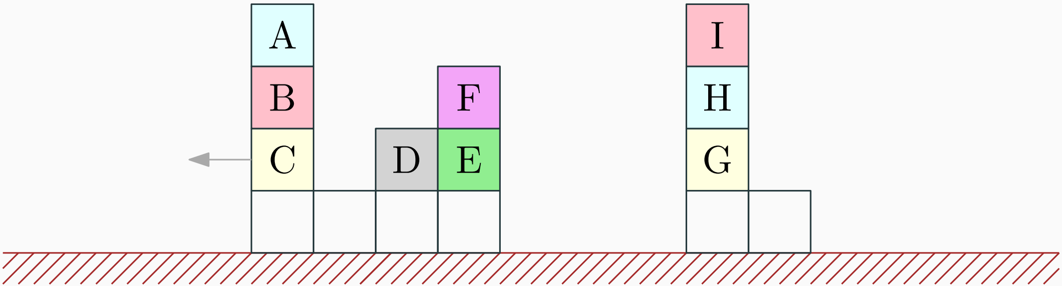


# B: Bulldozer

Problem Author: Mees de Vries

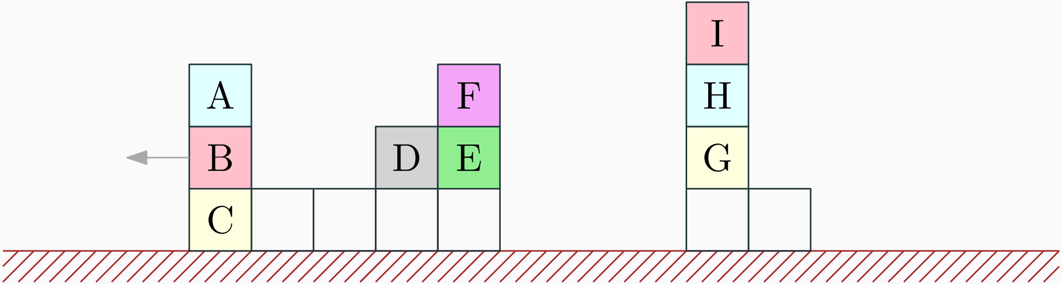


# B: Bulldozer
Problem Author: Mees de Vries



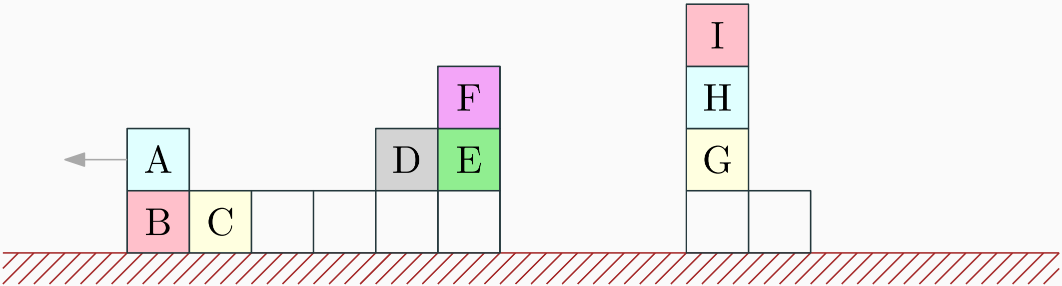
# B: Bulldozer

Problem Author: Mees de Vries



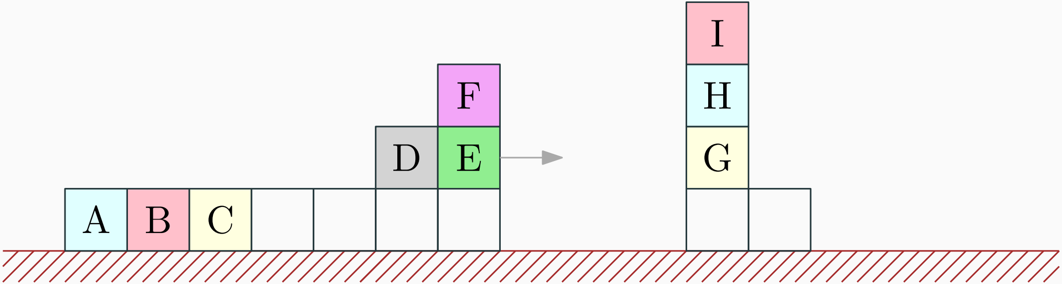
# B: Bulldozer

Problem Author: Mees de Vries



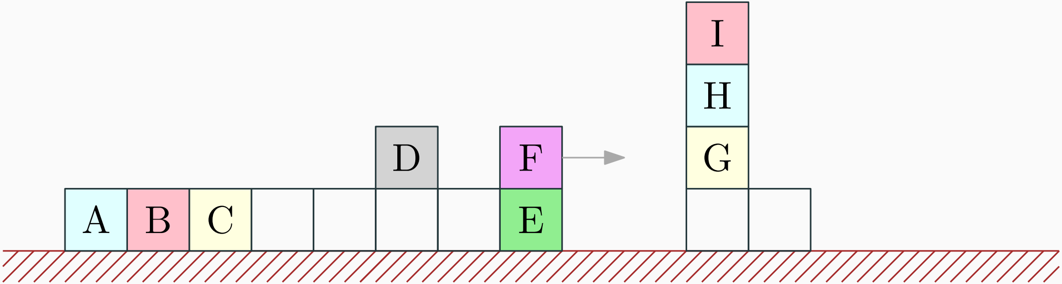
# B: Bulldozer

Problem Author: Mees de Vries



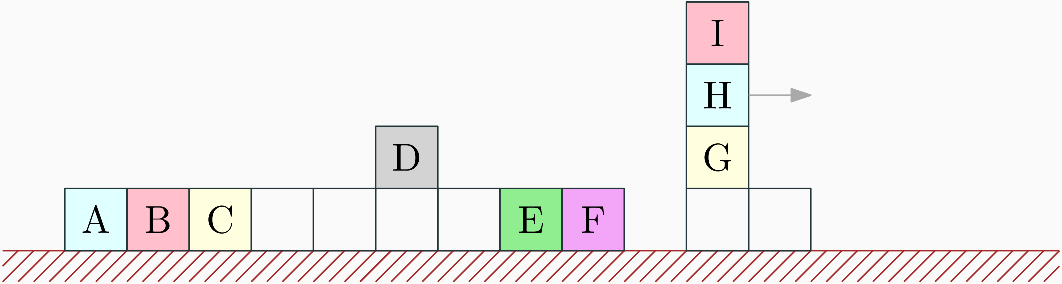
# B: Bulldozer

Problem Author: Mees de Vries



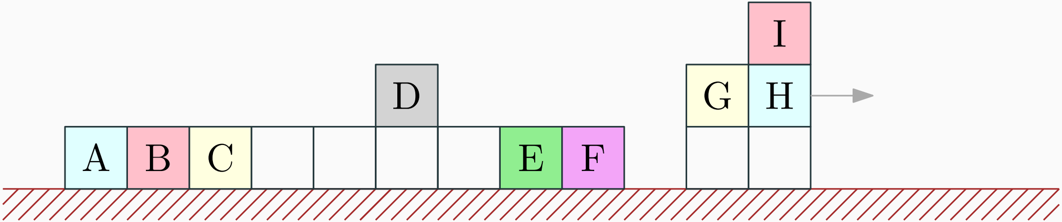
# B: Bulldozer

Problem Author: Mees de Vries



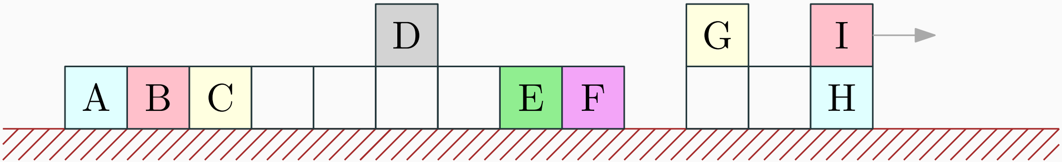
# B: Bulldozer

Problem Author: Mees de Vries



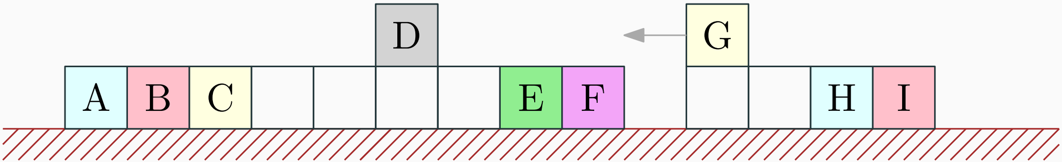
# B: Bulldozer

Problem Author: Mees de Vries



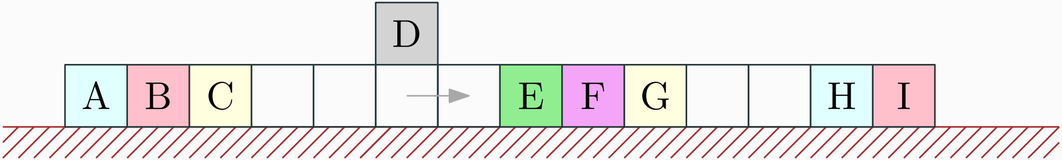
# B: Bulldozer

Problem Author: Mees de Vries



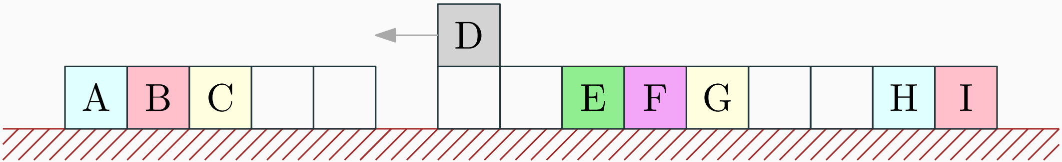
# B: Bulldozer

Problem Author: Mees de Vries



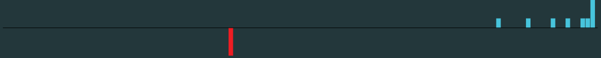
# B: Bulldozer

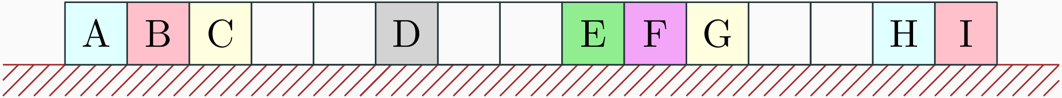
Problem Author: Mees de Vries



# B: Bulldozer

Problem Author: Mees de Vries



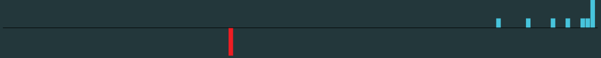


## Solution

- Make a weighted directed graph on the initial state of the blocks, with a start vertex on the far left and an end vertex on the far right. The shortest path will be the answer.
- For each empty stack S, find the block X that would end there when moving blocks from the left. Add an edge from X to S of cost K, the required number of moves for this.
- Similarly, find the block Y that would end at S when moving blocks from the right. Add an edge from S to Y of cost K.
- When block X ends in empty stack Y after K moves, all blocks in between are already levelled.
- Add an edge from the start vertex to the top of each stack: the cost of moving all in between blocks left.
- Add an edge from the bottom of each stack to the end vertex: the cost of moving all in between blocks right.
- For burying, add an edge between consecutive blocks of cost 2, but merge adjacent edges when possible to prevent adding $2 \cdot 10^{14}$ edges.

# B: Bulldozer

Problem Author: Mees de Vries



Statistics: 12 submissions, 0 + ? accepted

# C: Contest Struggles

Problem Author: Ragnar Groot Koerkamp

## Problem

For $n$ numbers between 0 and 100 you are given the average of all numbers ($d$), and the average of a subset of $k$ of those numbers ($s$). Compute the average of the remaining numbers.

## Solution

- The sum of all numbers is $d \cdot n$.
- So the sum of the remaining numbers is $d \cdot n - s \cdot k$.
- That parts contains $n - k$ numbers, so the average of those numbers is $(d \cdot n - s \cdot k)/(n - k)$.
- When the average is $< 0$ or $> 100$, print impossible.

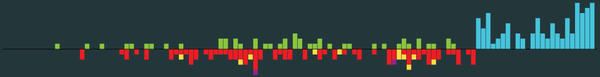
## Gotchas

- Precision issues, e.g. answers just below 0 or just above 100

Statistics: 180 submissions, 118 + ? accepted

# D: Dyson Circle

Problem Author: Mees de Vries

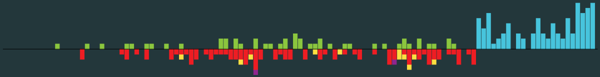


## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

# D: Dyson Circle

Problem Author: Mees de Vries

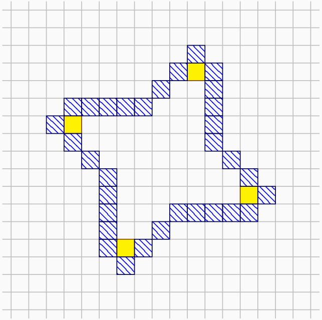


## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

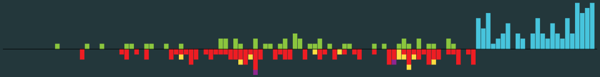
## Solution

- Let's look at the first sample.



# D: Dyson Circle

Problem Author: Mees de Vries

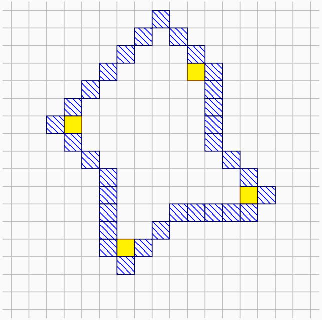


## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

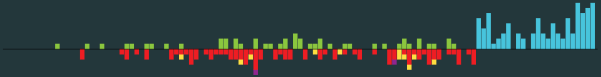
## Solution

- Let's look at the first sample.
- We might as well remove a "dent" in our Dyson circle.



# D: Dyson Circle

Problem Author: Mees de Vries

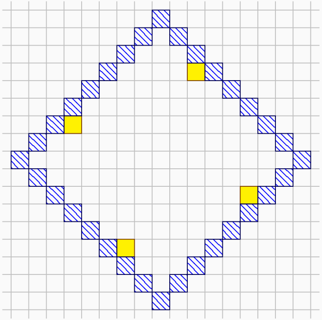


## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

## Solution

- Let's look at the first sample.
- We might as well remove a "dent" in our Dyson circle.
- In fact, we can do this with all dents.

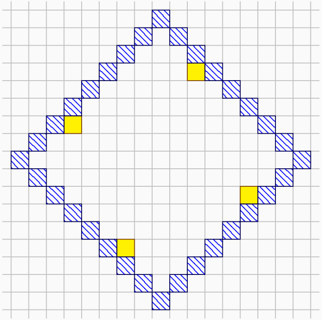


## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

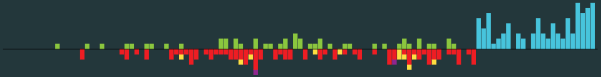
## Solution

- Let's look at the first sample.
- We might as well remove a "dent" in our Dyson circle.
- In fact, we can do this with all dents.
- In general, a rectangle with diagonal edges is *always* an optimal solution.



# D: Dyson Circle

Problem Author: Mees de Vries

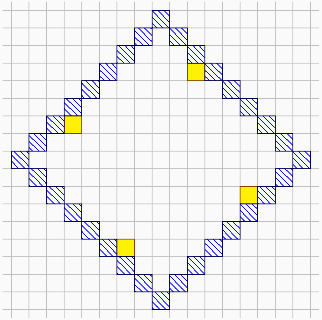


## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

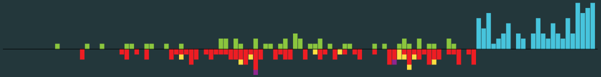
## Solution

- The only suns that matter are the four suns that touch the edges of the rectangle: the ones that maximize $x + y$, $x - y$, $-x + y$, $-x - y$.



# D: Dyson Circle

Problem Author: Mees de Vries



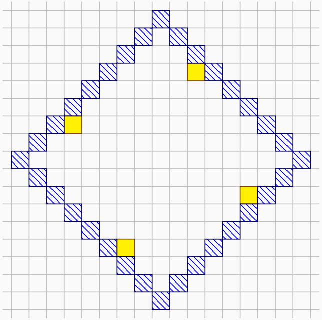
## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

## Solution

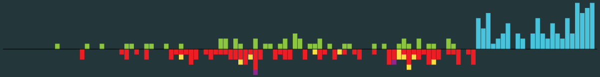
- The only suns that matter are the four suns that touch the edges of the rectangle: the ones that maximize $x+y$, $x-y$, $-x+y$, $-x-y$.
- So the general answer is

$$4 + \max_i(x_i + y_i) + \max_i(x_i - y_i) + \max_i(-x_i + y_i) + \max_i(-x_i - y_i).$$



# D: Dyson Circle

**Problem Author: Mees de Vries**



## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

# D: Dyson Circle

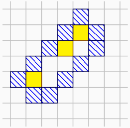
Problem Author: Mees de Vries

## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

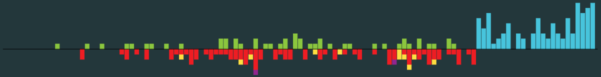
## Gotchas

- If all of the suns are on a diagonal, you need one additional square to make the inside a contiguous region.



# D: Dyson Circle

Problem Author: Mees de Vries

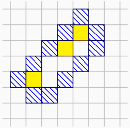


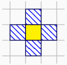
## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

## Gotchas

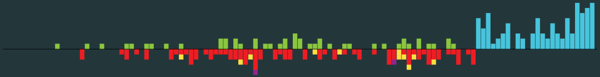
- If all of the suns are on a diagonal, you need one additional square to make the inside a contiguous region.
- However, if there is only one sun you do not need the additional square.





# D: Dyson Circle

Problem Author: Mees de Vries

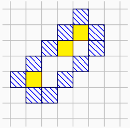


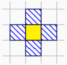
## Problem

Given some stars on a grid, encircle these with as few other grid points as possible.

## Gotchas

- If all of the suns are on a diagonal, you need one additional square to make the inside a contiguous region.
- However, if there is only one sun you do not need the additional square.

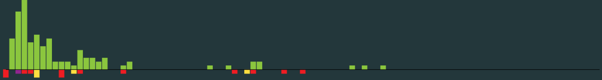




Statistics: 248 submissions, 48 accepted, 99 unknown

# A: Another Eruption

Problem Author: Jeroen Bransen, Bjarki Ágúst Guðmundsson

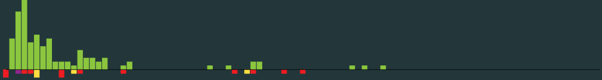


## Problem

We want to put a barrier tape around the border of a circular gas cloud. The area of the gas cloud in metres$^2$ is already known. Tell us its perimeter.

# A: Another Eruption

Problem Author: Jeroen Bransen, Bjarki Ágúst Guðmundsson



## Solution

- The area $a$ of a circle with radius $r$ is given by $\pi r^2$.
- The perimeter $p$ of such a circle is $2\pi r$.
- Because $a = \pi r^2$, we know $r = \sqrt{\frac{a}{\pi}}$.
- Hence $p = 2\pi\sqrt{\frac{a}{\pi}} = \sqrt{4\pi a}$.

## Gotchas

- Remember to print with high-precision:
    - C++: `cout.precision(12)` or `printf("%.9f\n", p)`
    - Python: `"{:.9f}".format(p)`
    - Java: `System.out.printfln("%.9f\n", p)`
- Use `long` or `double` to read the input, $10^{18} > 2^{31}$

Statistics: 123 submissions, 106 accepted

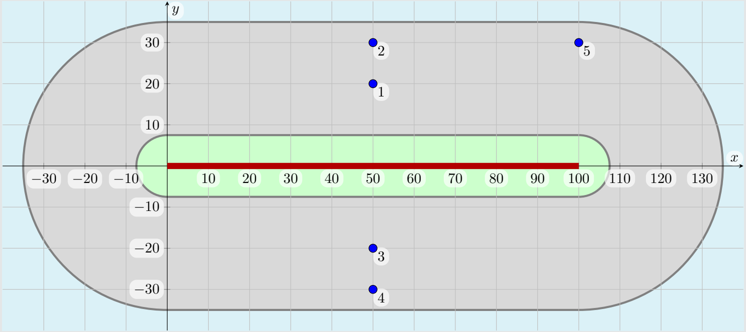
# F: Flatland Olympics

Problem Author: Harry Smit

## Problem

Given a line segment $s$ and a set of $n$ points $p_1, \ldots, p_n$. Find the number of pairs of points $p_i, p_j$ $(i < j)$ such that both points lie on the same side of $s$ and the line through $p_i$ and $p_j$ intersects $s$.

## Example

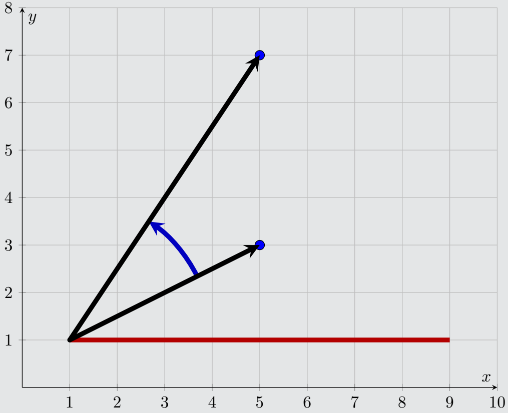


## Observation

- Observe how the relation of two points changes while moving from one end to the other of the line segment $s$:

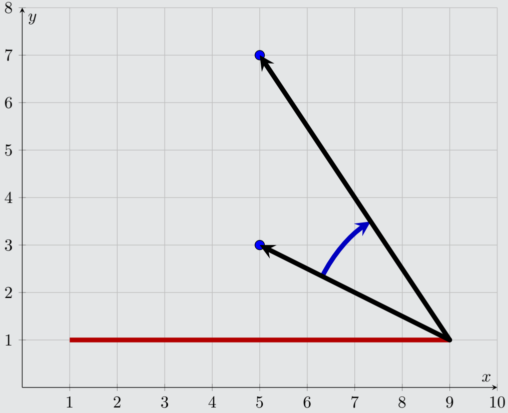
## Observation

- Observe how the relation of two points changes while moving from one end to the other of the line segment $s$:



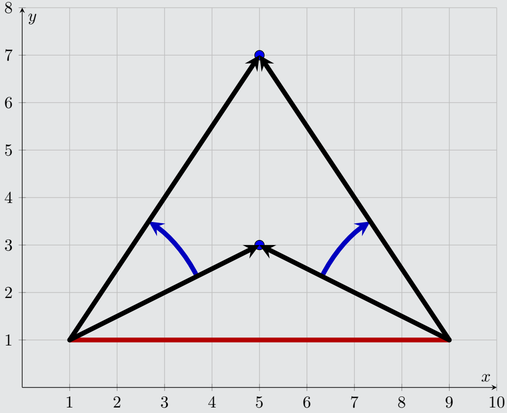
## Observation

- Observe how the relation of two points changes while moving from one end to the other of the line segment $s$:



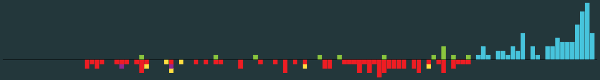
## Observation

- Observe how the relation of two points changes while moving from one end to the other of the line segment $s$:



# F: Flatland Olympics

Problem Author: Harry Smit

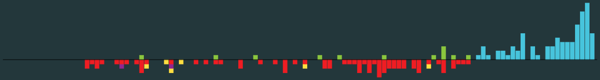


## Solution

- Separate the points above and below $s$ in two different sets.

# F: Flatland Olympics

Problem Author: Harry Smit



## Solution

- Separate the points above and below $s$ in two different sets.
- For each set:
  - Sort the points around the *start* of $s$.
  - Sort the points around the *end* of $s$.
  - A pair of points has to be counted if their order in these two sequences differ.

# F: Flatland Olympics

Problem Author: Harry Smit

## Solution

- Separate the points above and below $s$ in two different sets.
- For each set:
  - Sort the points around the *start* of $s$.
  - Sort the points around the *end* of $s$.
  - A pair of points has to be counted if their order in these two sequences differ.
- We need to find the number of *inversions* between two permutations.
- This can be done in $\mathcal{O}(n \log(n))$.

# F: Flatland Olympics

Problem Author: Harry Smit

## Solution

- Separate the points above and below $s$ in two different sets.
- For each set:
  - Sort the points around the *start* of $s$.
  - Sort the points around the *end* of $s$.
  - A pair of points has to be counted if their order in these two sequences differ.
- We need to find the number of *inversions* between two permutations.
- This can be done in $\mathcal{O}(n \log(n))$.

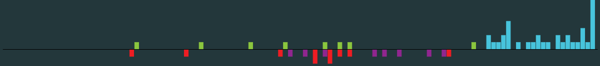
## Gotcha

- Points lying along the line through $s$.
- Multiple points collinear with the start or the end of $s$.

Statistics: 179 submissions, 12 accepted, 86 unknown

# G: Great Expectations

Problem Author: Mees de Vries



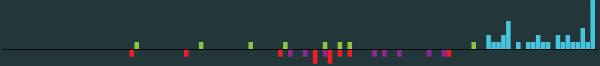
## Problem

Determine the most efficient method to break the record in a speedrun. You may reset at any point.

## Insights

During a run, you have $r - n - 1$ time margin to make errors.

Optimally, the only place where you reset is immediately after failing a trick.



## Solution attempt

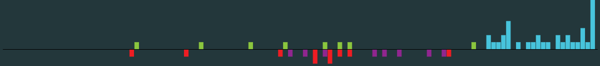
- Use dynamic programming!
- $DP[i,j] :=$ the expected time until a record when you are just before trick $i$ and have used $j$ margin for error. We are interested in $DP[0,0]$.
- When you complete trick $i$, the rest of the run takes $(t_{i+1} - t_i) + DP[i+1,j]$ time.
- When you fail the trick, you either reset (taking $DP[0,0]$ time) or continue (taking $d_i + (t_{i+1} - t_i) + DP[i+1, j+d_i]$ time).
- This gives a DP relation:

$$DP[i,j] = \begin{array}{rcl} p_i & \cdot & ((t_{i+1} - t_i) + DP[i+1,j]) + \\ (1 - p_i) & \cdot & \min(DP[0,0], d_i + (t_{i+1} - t_i) + DP[i+1, j+d_i]) \end{array}$$

- We can use $DP[m][j] = 0$ as the base cases for the DP.

# G: Great Expectations

Problem Author: Mees de Vries



## Catch

We now have a DP relation, but we need to know $DP[0,0]$ in order to use it.

## Solution

- Consider making some guess $P$ for the value of $DP[0,0]$. We can use this value to fill the DP table.
- When the resulting $DP[0,0]$ is larger than $P$, the guess was too low. When $DP[0,0]$ is smaller than $P$, the guess was too high.
- Use binary search to determine the optimal value of $P$, and thus the actual value of $DP[0,0]$.

Statistics: 61 submissions, 8 + ? accepted

# H: Heating Up

Problem Author: Alexander Dietsch

## Problem

Given a pizza with many slices, each having its own spiciness level. Eating a slice with a certain spiciness is only possible if you have enough tolerance, and it increases this tolerance by the spiciness level of the slice.

You are allowed to start at any slice but after every slice, you must continue with one of the neighbouring slices. Which initial minimal tolerance is needed to finish the pizza.

### Solution

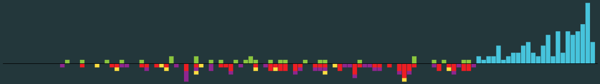
- Problem can be solved with binary search. (If tolerance $x$ is enough, $x+1$ works as well)

### Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?

# H: Heating Up

Problem Author: Alexander Dietsch



## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice $=$ 1 element.

## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice $=$ 1 element.
- Visit all elements; on a visit:

# H: Heating Up

Problem Author: Alexander Dietsch

## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice = 1 element.
- Visit all elements; on a visit:
  - Check if the initial tolerance is high enough to finish the element.

# H: Heating Up

Problem Author: Alexander Dietsch

## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice $=$ 1 element.
- Visit all elements; on a visit:
  - Check if the initial tolerance is high enough to finish the element.
  - If so, check if the resulting tolerance is enough to finish a neighbouring element.

# H: Heating Up

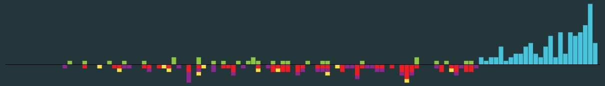
Problem Author: Alexander Dietsch

## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice $=$ 1 element.
- Visit all elements; on a visit:
  - Check if the initial tolerance is high enough to finish the element.
  - If so, check if the resulting tolerance is enough to finish a neighbouring element.
  - If that is the case, merge the elements. The spiciness level to finish the new element is the minimum, the increase in tolerance is the sum of both elements.

# H: Heating Up

Problem Author: Alexander Dietsch

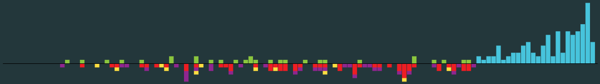


## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice $=$ 1 element.
- Visit all elements; on a visit:
  - Check if the initial tolerance is high enough to finish the element.
  - If so, check if the resulting tolerance is enough to finish a neighbouring element.
  - If that is the case, merge the elements. The spiciness level to finish the new element is the minimum, the increase in tolerance is the sum of both elements.
- If the linked list can be merged into a single element, the initial tolerance is enough to finish the pizza.

# H: Heating Up

Problem Author: Alexander Dietsch



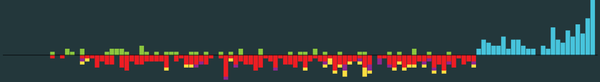
## Solution

- Problem can be solved with binary search. (If tolerance $x$ is enough, $x + 1$ works as well)
- New problem: Does tolerance $x$ suffice to eat the whole pizza?
- Use a cyclic linked list, each element holds the spiciness level to finish the element and the increase in tolerance it gives. Initially 1 slice $=$ 1 element.
- Visit all elements; on a visit:
    - Check if the initial tolerance is high enough to finish the element.
    - If so, check if the resulting tolerance is enough to finish a neighbouring element.
    - If that is the case, merge the elements. The spiciness level to finish the new element is the minimum, the increase in tolerance is the sum of both elements.
- If the linked list can be merged into a single element, the initial tolerance is enough to finish the pizza.

Statistics: 252 submissions, 29 accepted, 124 unknown

# F: Flight Collision

Problem Author: Jorke de Vlas

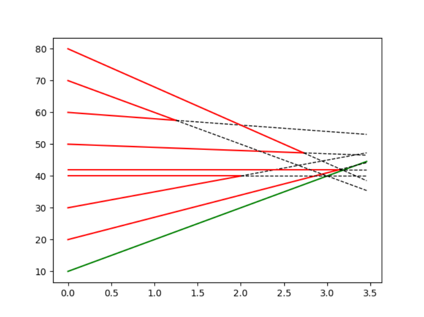


## Problem

Some drones are flying along a straight line at constant speed. Simulate the crashes and report the survivors.

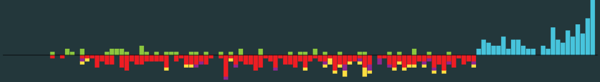
## Insight

At any moment, the next crash is going to be between two adjacent drones.



# F: Flight Collision

Problem Author: Jorke de Vlas



## Solution

- Maintain a set of potential crash events, sorted by time.
- The crash times can be found by solving linear equations.
- When processing a crash, add a new event for the two drones that become adjacent.
- Time complexity: $\mathcal{O}(n \log n)$.

## Gotchas

- Use fractions or `long double` to avoid precision errors.
- Only consider crashes at times $t > 0$.

Statistics: 421 submissions, 46 + ? accepted

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is yes.

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is yes.
- Naïve solution:

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is yes.
- Naïve solution:
  - Keep an array of 720 booleans, one for each meridian and half-meridian.

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is yes.
- Naïve solution:
  - Keep an array of 720 booleans, one for each meridian and half-meridian.
  - When travelling to a new longitude, loop over the array and set the visited longitudes to true.

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is `yes`.
- Naïve solution:
  - Keep an array of 720 booleans, one for each meridian and half-meridian.
  - When travelling to a new longitude, loop over the array and set the visited longitudes to true.
  - Finally, output `yes` if every element of the array is `true`, and `no` otherwise.

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is `yes`.
- Naïve solution:
  - Keep an array of 720 booleans, one for each meridian and half-meridian.
  - When travelling to a new longitude, loop over the array and set the visited longitudes to true.
  - Finally, output `yes` if every element of the array is `true`, and `no` otherwise.
- This naïve solution is correct!

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is `yes`.
- Naïve solution:
  - Keep an array of 720 booleans, one for each meridian and half-meridian.
  - When travelling to a new longitude, loop over the array and set the visited longitudes to true.
  - Finally, output `yes` if every element of the array is `true`, and `no` otherwise.
- This naïve solution is correct!
- Pitfalls: be careful to correctly operate on the circular array.

# J: Jet Set

Problem Author: Paul Wild

## Problem

Given a list of stops on a trip, determine whether it passes through every meridian.

## Solution

- Observations:
  - You can ignore the latitudes – they do not matter.
  - If the longitude ever changes by 180 in a single flight, the trip goes over one of the poles, so the answer is `yes`.
- Naïve solution:
  - Keep an array of 720 booleans, one for each meridian and half-meridian.
  - When travelling to a new longitude, loop over the array and set the visited longitudes to true.
  - Finally, output `yes` if every element of the array is `true`, and `no` otherwise.
- This naïve solution is correct!
- Pitfalls: be careful to correctly operate on the circular array.

Statistics: 342 submissions, 81 accepted, 74 unknown

# J: Jet Set

Problem Author: Paul Wild

## Edge case


Don't forget the edge case of going around for 359° degrees and then turning around!

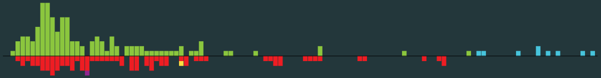
## Edge case

```
--- Original
+++ New
@@ @@
         cout <<setprecision(1) << fixed;
         double dres = res/2.0;
         double unfix = dres >= M/2 ? dres -M : dres;
-        cout << unfix << "\n";
+        cout << "no " <<  unfix << "\n";
     }
 }
```

Please read the output section carefully.

# K: Knitpicking

Problem Author: Pehr Söderman

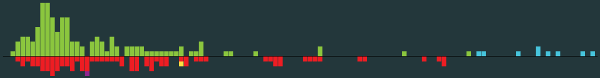


## Problem

Given a drawer full of socks, compute how many you need to pick to be guaranteed to have a pair.

# K: Knitpicking

Problem Author: Pehr Söderman



## Problem

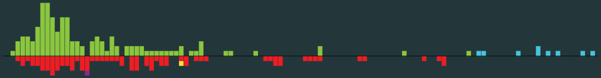
Given a drawer full of socks, compute how many you need to pick to be guaranteed to have a pair.

## Solution

- Count the number of socks you can pick *without* a pair, then add 1 at the end.

# K: Knitpicking

Problem Author: Pehr Söderman



## Problem

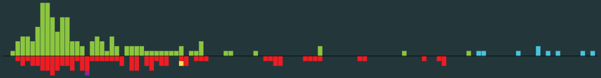
Given a drawer full of socks, compute how many you need to pick to be guaranteed to have a pair.

## Solution

- Count the number of socks you can pick *without* a pair, then add 1 at the end.
- For every type of socks you can pick $\max(\texttt{left}, \texttt{right}, 1)$ socks: all socks of one side, or 1 any sock.

# K: Knitpicking

Problem Author: Pehr Söderman



## Problem

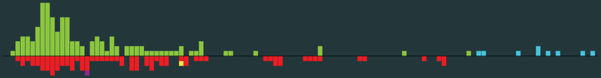
Given a drawer full of socks, compute how many you need to pick to be guaranteed to have a pair.

## Solution

- Count the number of socks you can pick *without* a pair, then add 1 at the end.
- For every type of socks you can pick $\max(\texttt{left}, \texttt{right}, 1)$ socks: all socks of one side, or 1 any sock.
- Remember to output `impossible` when every sock type only has `left` socks, `right` socks, or a single `any` sock.

# K: Knitpicking

Problem Author: Pehr Söderman



## Problem

Given a drawer full of socks, compute how many you need to pick to be guaranteed to have a pair.

## Solution

- Count the number of socks you can pick *without* a pair, then add 1 at the end.
- For every type of socks you can pick $\max(\texttt{left}, \texttt{right}, 1)$ socks: all socks of one side, or 1 any sock.
- Remember to output `impossible` when every sock type only has `left` socks, `right` socks, or a single `any` sock.

Statistics: 218 submissions, 126 accepted, 9 unknown

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Problem

Given is a list of $n$ shirts. We choose $k$ integers $l_1, \ldots, l_k$ uniformly at random and then randomly permute the first $l_j$ shirts for $j \in \{1, \ldots, k\}$. What is the expected position of the shirt that started at position $i$ (1-based)?

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Problem

Given is a list of $n$ shirts. We choose $k$ integers $l_1, \ldots, l_k$ uniformly at random and then randomly permute the first $l_j$ shirts for $j \in \{1, \ldots, k\}$. What is the expected position of the shirt that started at position $i$ (1-based)?

## First idea

- Calculate the probability $p_a$ that your lucky shirt ends up at position $a$ for all $a \in \{1, \ldots, n\}$.

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Problem

Given is a list of $n$ shirts. We choose $k$ integers $l_1, \ldots, l_k$ uniformly at random and then randomly permute the first $l_j$ shirts for $j \in \{1, \ldots, k\}$. What is the expected position of the shirt that started at position $i$ (1-based)?

## First idea

- Calculate the probability $p_a$ that your lucky shirt ends up at position $a$ for all $a \in \{1, \ldots, n\}$.
- The answer is

$$\sum_{a=1}^{n} a \cdot p_a.$$

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Problem

Given is a list of $n$ shirts. We choose $k$ integers $l_1, \ldots, l_k$ uniformly at random and then randomly permute the first $l_j$ shirts for $j \in \{1, \ldots, k\}$. What is the expected position of the shirt that started at position $i$ (1-based)?

## First idea

- Calculate the probability $p_a$ that your lucky shirt ends up at position $a$ for all $a \in \{1, \ldots, n\}$.
- The answer is

$$\sum_{a=1}^{n} a \cdot p_a.$$

- However, $p_a$ does not have a nice formula.

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.
  - This happens exactly when $M < i$.

### Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.
  - This happens exactly when $M < i$.
  - The (expected) position of the shirt is $i$.

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.
  - This happens exactly when $M < i$.
  - The (expected) position of the shirt is $i$.
- Case 2: the shirt is shuffled at least once.

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.
  - This happens exactly when $M < i$.
  - The (expected) position of the shirt is $i$.
- Case 2: the shirt is shuffled at least once.
  - This happens exactly when $M \geq i$.

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.
  - This happens exactly when $M < i$.
  - The (expected) position of the shirt is $i$.
- Case 2: the shirt is shuffled at least once.
  - This happens exactly when $M \geq i$.
  - You cannot distinguish the lucky shirt from any of the other first $M$ shirts

## Solution (1/2)

- Key observation: once the lucky shirt is shuffled, its location is uniform among the shuffled shirts.
- Only $M := \max_j l_j$ is relevant! We distinguish two simple cases.
- Case 1: the shirt never moves during the process.
  - This happens exactly when $M < i$.
  - The (expected) position of the shirt is $i$.
- Case 2: the shirt is shuffled at least once.
  - This happens exactly when $M \geq i$.
  - You cannot distinguish the lucky shirt from any of the other first $M$ shirts
  - The (expected) position of the shirt is $(M+1)/2$.

## Solution (2/2)

- Thus the answer equals
$$i \cdot \mathbb{P}(M < i) + \sum_{a=i}^{n} \frac{a+1}{2} \cdot \mathbb{P}(M = a).$$

## Solution (2/2)

- Thus the answer equals

$$i \cdot \mathbb{P}(M < i) + \sum_{a=i}^{n} \frac{a+1}{2} \cdot \mathbb{P}(M = a).$$

- As the $l_j$ are chosen uniformly at random (and independent of one another),

## Solution (2/2)

- Thus the answer equals

$$i \cdot \mathbb{P}(M < i) + \sum_{a=i}^{n} \frac{a+1}{2} \cdot \mathbb{P}(M = a).$$

- As the $l_j$ are chosen uniformly at random (and independent of one another),

# L: Lucky Shirt

Problem Author: Ragnar Groot Koerkamp

## Solution (2/2)

- Thus the answer equals
$$i \cdot \mathbb{P}(M < i) + \sum_{a=i}^{n} \frac{a+1}{2} \cdot \mathbb{P}(M = a).$$
- As the $I_j$ are chosen uniformly at random (and independent of one another),
$$\mathbb{P}(M < i) = \left(\frac{i-1}{n}\right)^k, \text{ and}$$
$$\mathbb{P}(M = a) = \mathbb{P}(M < a+1) - \mathbb{P}(M < a) = \left(\frac{a}{n}\right)^k - \left(\frac{a-1}{n}\right)^k.$$

Statistics: 30 submissions, 1 accepted, 25 unknown